

# Python code

```
1
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5
6 # ## **Question 1**
7
8
9 def func1(X):
10     p1, p2 = 0.5, 0.5
11     m1 = np.array([[1, -1]])
12     m2 = np.array([[2, 3]])
13     sigma1 = np.array([[2, 3], [3, 6.5]])
14     sigma2 = np.array([[2, 3], [3, 6.5]])
15     res = []
16     for x in X:
17         px1 = p1 / (np.linalg.det(sigma1)**0.5) * np.exp(-0.5 * (x - m1) @
np.linalg.inv(sigma1) @ (x - m1).T)
18         px2 = p2 / (np.linalg.det(sigma2)**0.5) * np.exp(-0.5 * (x - m2) @
np.linalg.inv(sigma2) @ (x - m2).T)
19         res.append(px1 / px2)
20     return np.array(res)
21
22 def func2(X):
23     p1, p2 = 0.1, 0.9
24     m1 = np.array([[1, -1]])
25     m2 = np.array([[2, 3]])
26     sigma1 = np.array([[2, 3], [3, 6.5]])
27     sigma2 = np.array([[2, 3], [3, 6.5]])
28     res = []
29     for x in X:
30         px1 = p1 / (np.linalg.det(sigma1)**0.5) * np.exp(-0.5 * (x - m1) @
np.linalg.inv(sigma1) @ (x - m1).T)
31         px2 = p2 / (np.linalg.det(sigma2)**0.5) * np.exp(-0.5 * (x - m2) @
np.linalg.inv(sigma2) @ (x - m2).T)
32         res.append(px1 / px2)
33     return np.array(res)
34
35
36 def plot_decision_boundary(training, label_train, func):
37     # Total number of classes
38     classes = np.unique(label_train)
39     nclass = len(classes)
40
41     class_names = []
42     for c in classes:
43         class_names.append('class ' + str(int(c)))
44
45     # Set the feature range for plotting
46     max_x1 = np.ceil(np.max(training[:, 0])) + 1.0
47     min_x1 = np.floor(np.min(training[:, 0])) - 1.0
48     max_x2 = np.ceil(np.max(training[:, 1])) + 1.0
```

```

49     min_x2 = np.floor(np.min(training[:, 1])) - 1.0
50
51     xrange = (min_x1, max_x1)
52     yrange = (min_x2, max_x2)
53
54     # step size for how finely you want to visualize the decision boundary.
55     inc = 0.05
56
57     # generate grid coordinates. This will be the basis of the decision
58     boundary visualization.
59     (x1, x2) = np.meshgrid(np.arange(xrange[0], xrange[1] + inc / 100,
60                                     inc),
61                             np.arange(yrange[0], yrange[1] + inc / 100,
62                                     inc))
63
64     # size of the (x1, x2) image, which will also be the size of the
65     # decision boundary image that is used as the plot background.
66     image_size = x1.shape
67     # make (x1, x2) pairs as a bunch of row vectors.
68     grid_2d = np.hstack((x1.reshape(x1.shape[0] * x1.shape[1], 1,
69                                     order='F'),
70                           x2.reshape(x2.shape[0] * x2.shape[1], 1,
71                                     order='F'))))
72
73     pred_label = np.where(func(grid_2d) > 1, 1, 2)
74     # reshape the idx (which contains the class label) into an image.
75     decision_map = pred_label.reshape(image_size, order='F')
76
77     # create fig
78     fig, ax = plt.subplots()
79     ax.imshow(decision_map, vmin=np.min(classes), vmax=9, cmap='Pastel1',
80              extent=[xrange[0], xrange[1], yrange[0], yrange[1]],
81              origin='lower')
82
83     # plot the class training data.
84     data_point_styles = ['rx', 'bo', 'g*']
85     for i in range(nclass):
86         ax.plot(training[label_train == classes[i], 0],
87                 training[label_train == classes[i], 1],
88                 data_point_styles[int(classes[i]) - 1],
89                 label=class_names[i])
90     ax.legend()
91
92     plt.tight_layout()
93     plt.show()
94
95     return fig
96
97
98 training = np.array([[4, -1], [1, 3]])
99 label_train = [1, 2]
100 fig = plot_decision_boundary(training, label_train, func1)
101 fig.savefig("./figs/1d1.png", dpi=200)

```

```

99 training = np.array([[4, -1], [1, 3]])
100 label_train = [1, 2]
101 fig = plot_decision_boundary(training, label_train, func2)

```

```

102 fig.savefig("./figs/1d2.png", dpi=200)
103
104
105 # ## **Question 2**
106
107
108 def phi(x, h):
109     x1 = np.array([0, 0.4, 0.9, 1.0, 6.0, 8.0])
110     x2 = np.array([2.0, 4.0, 4.5, 5.0, 5.8, 6.7, 7.0])
111     x1 = x - x1
112     x2 = x - x2
113     return np.where((-h <= x1) & (x1 <= h), 1, 0), np.where((-h <= x2) &
114 (x2 <= h), 1, 0)
115
116 def density(x, h):
117     res1, res2 = phi(x, h)
118     n1 = res1.shape[0]
119     n2 = res2.shape[0]
120     return 1/n1 * np.sum(res1), 1/n2 * np.sum(res2)
121
122 x = np.linspace(-2, 10, 1201)
123 h = 0.5
124 P1 = []
125 P2 = []
126 for i in range(len(x)):
127     p1, p2 = density(x[i], h)
128     P1.append(p1)
129     P2.append(p2)
130
131 _ = plt.plot(x, P1, label='P(x|S_1)')
132 _ = plt.plot(x, P2, label='P(x|S_2)')
133 _ = plt.legend()
134 _ = plt.xlabel("X")
135 _ = plt.ylabel("Prob")
136 _ = plt.title("Kernel Density Estimation (h={})".format(h))
137 plt.savefig("./figs/kde_{}.png".format(h))
138
139 x = np.linspace(-2, 10, 1201)
140 h = 1
141 P1 = []
142 P2 = []
143 for i in range(len(x)):
144     p1, p2 = density(x[i], h)
145     P1.append(p1)
146     P2.append(p2)
147
148 _ = plt.plot(x, P1, label='P(x|S_1)')
149 _ = plt.plot(x, P2, label='P(x|S_2)')
150 _ = plt.legend()
151 _ = plt.xlabel("X")
152 _ = plt.ylabel("Prob")
153 _ = plt.title("Kernel Density Estimation (h={})".format(h))
154 plt.savefig("./figs/kde_{}.png".format(h))
155
156
157 x = np.linspace(-5, 15, 2001)
158 h = 2

```

```

159 P1 = []
160 P2 = []
161 for i in range(len(x)):
162     p1, p2 = density(x[i], h)
163     P1.append(p1)
164     P2.append(p2)
165
166 _ = plt.plot(x, P1, label='P(x|S_1)')
167 _ = plt.plot(x, P2, label='P(x|S_2)')
168 _ = plt.legend()
169 _ = plt.xlabel("x")
170 _ = plt.ylabel("Prob")
171 _ = plt.title("Kernel Density Estimation (h={})".format(h))
172 plt.savefig("./figs/kde_{}.png".format(h))
173
174
175 def decision_rule(x, h):
176     res1, res2 = phi(x, h)
177     return int(np.sum(res1) / np.sum(res2) > 1)
178
179 x = np.linspace(-2, 10, 1201)
180 h = 0.5
181 y = []
182 for i in range(len(x)):
183     y.append(decision_rule(x[i], h))
184
185 _ = plt.scatter(x, y, s=0.1)
186 _ = plt.yticks([0, 1])
187 _ = plt.xlabel("x")
188 _ = plt.ylabel("class")
189 _ = plt.title("Decision Boundary")
190 plt.savefig("./figs/dc_h_{}.png".format(h))
191 plt.show()
192
193 x = np.linspace(-2, 10, 1201)
194 h = 1
195 y = []
196 for i in range(len(x)):
197     y.append(decision_rule(x[i], h))
198
199 _ = plt.scatter(x, y, s=0.1)
200 _ = plt.yticks([0, 1])
201 _ = plt.xlabel("x")
202 _ = plt.ylabel("class")
203 _ = plt.title("Decision Boundary")
204 plt.savefig("./figs/dc_h_{}.png".format(h))
205 plt.show()
206
207 x = np.linspace(-5, 15, 2001)
208 h = 2
209 y = []
210 for i in range(len(x)):
211     y.append(decision_rule(x[i], h))
212
213 _ = plt.scatter(x, y, s=0.1)
214 _ = plt.yticks([0, 1])
215 _ = plt.xlabel("x")
216 _ = plt.ylabel("class")

```

```

217 _ = plt.title("Decision Boundary")
218 plt.savefig("./figs/dc_h_{}.png".format(h))
219 plt.show()
220
221
222 # ## **Question 3**
223
224
225 x = np.array([-0.9, -0.7, -0.5, -0.3, -0.1, 0.1, 0.3, 0.5, 0.7, 0.9])
226 y = x**2
227 x, y
228
229
230 x1 = 0.0
231 d = np.abs(x - x1)
232 idx = np.argsort(d)[:4]
233 dmax = np.sort(d)[5]
234 weight = 1 - d / dmax
235 y1 = np.sum(weight[idx] * y[idx]) / np.sum(weight[idx])
236 weight
237 y1
238
239
240 x2 = 0.4
241 d = np.abs(x - x2)
242 idx = np.argsort(d)[:4]
243 dmax = np.sort(d)[5]
244 weight = 1 - d / dmax
245 y2 = np.sum(weight[idx] * y[idx]) / np.sum(weight[idx])
246 weight
247 y2
248
249
250 0.5*((y1-0)**2+(y2-0.16)**2)
251
252
253

```